ActiveMath: An Intelligent Tutoring System for Mathematics

Erica Melis and Jörg Siekmann

German Research Institute for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg, 66123 Saarbrücken, Germany
phone: +49 681 302 4629, fax: +49 681 302 2235

Abstract. ActiveMath is a web-based intelligent tutoring system for mathematics. This article presents the technical and pedagogical goals of ActiveMath, its principles of design and architecture, its knowledge representation, and its adaptive behavior. In particular, we concentrate on those features that rely on AI-techniques.

1 Introduction

Intelligent tutoring systems (ITSs) have been researched in AI now for several decades. With the enormous development and increasing availability of the Internet, the application of web-based learning systems becomes more likely and realistic and research for intelligent features receives more attention than before. As a result, a number of new ITS have been developed over the last five years, among them ActiveMath, a web-based, adaptive learning environment for mathematics.

These systems strive for improving long-distance learning, for complementing traditional classroom teaching, and for supporting individual and life-long learning. Web-based systems are available on central servers and allow a user to learn in her own environment and whenever it is appropriate for her.

Intelligent tutoring systems are a great field of application for AI-techniques. In a nutshell, our research for ActiveMath has used and further developed results in

- problem solving
- rule-based systems
- knowledge representation
- user modeling **?**
- adaptive systems and adaptive hyper-media
- diagnosis.

Learning environments have to meet realistic and complex needs rather than being a specific research system for specific demonstrations such as the famous blocksworld. Therefore, we point out important pedagogical and technical goals that our research for ActiveMath had to satisfy.

Pedagogical Goals

ActiveMath' design aims at supporting truly interactive, exploratory learning and assumes the student to be responsible for her learning to some extent. Therefore, a relative freedom for navigating through a course and for learning choices is given and by default, the student model is scrutable, i.e., inspectable and modifiable. Moreover, dependencies of learning objects can be inspected in a dictionary to help the student to learn the overall picture of a domain (e.g., analysis) and also the dependencies of concepts.

Several dimensions of adaptivity to the student and her learning context improve the learner's motivation and performance. Most previous intelligent tutor systems did not rely on an adaptive choice of content. A reason might be that the envisioned use was mostly in schools, where traditionally every student learns the same concepts for the same use. In colleges and universities, however, the same subject is already taught differently for different groups of users and in different contexts, e.g., statistics has to be taught differently for students of mathematics, for economics, or medicine. Therefore, the adaptive choice of content to be presented as well as examples and exercises is pivotal. In addition, an adaptation of examples and exercises to the student's capabilities is highly desirable in order to keep the learner in the zone of proximal development [13] rather than overtax or undertax her.

Moreover, web-based systems can be used in several learning contexts, e.g., long-distance learning, homework, and teacher-assisted learning. Personalization is required in all of them because even for teacher-assisted learning in a computer-free classroom with, say, 30 students and one teacher individualized learning is impossible. ActiveMath's current version provides adaptive content, adaptive presentation features, and adaptive appearance.

Technical Goals

Building quality hyper-media content is a time-consuming and costly process, hence the content should be *reusable* in different contexts. However, most of today's interactive textbooks consist of a collection of predefined

documents, typically canned HTML pages and multimedia animations. This situation makes a reuse in other contexts and a re-combination of the encoded knowledge impossible and inhibits a radical adaptation of course presentation and content to the user's needs.
ActiveMath' knowledge representation contributes to re-usability and interoperability. In particular, it is compliant with the emerging knowledge representation and communication standards such as Dublin Core, OpenMath, MathML, and LOM[1]. Some of the buzzwords here are metadata, ontological XML, (OMDoc [8], and standardized content packaging. Such features of knowledge representations will ensure a long-term employment of the new technologies in browsers and other devices. In order to use the potential power of existing web-based technology e-Learning systems need an open architecture to integrate and connect to new components including student management systems such as WebCT, assessment tools, collaboration tools, and problem solving tools.

Organization of the Article This article provides an overview of the current ActiveMath system. It describes some main features in more detail, in particular, the architecture and its components, the knowledge representation, the student model and the adaptation based on the information from the student model.

2 Architecture

The architecture of ActiveMath, as sketched in Figure 1, strictly realizes the principle of separation of (declarative) knowledge and functionalities as well as the separation of different kinds of knowledge. For instance, pedagogical knowledge is stored in a pedagogical rule base, the educational content is stored in MBase, and the knowledge about the user is stored in the student model. This principle has proved valuable in many AI-applications and eases modifications as well as configurability and reuse of the system.

ActiveMath has a client-server architecture whose client can be restricted to a browser. This architecture serves not only the openness but also the *platform independence*. On the client side, a browser – netscape higher than 6, Mozilla, or IE with MathPlayer – is sufficient to work with ActiveMath. On the server-side components of ActiveMath have been deliberately designed in a *modular* way in order to guarantee exchangeability and robustness.

[1] http://ltsc.ieee.org/wg12/

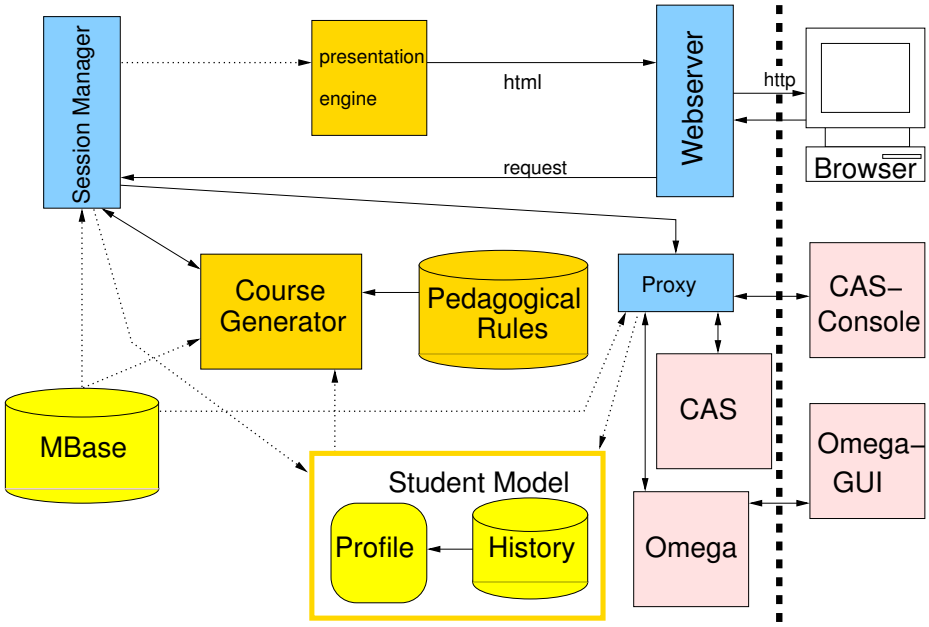

Fig. 1. Architecture of ActiveMath

When the user has chosen her goal concepts and learning scenario, the session manager sends a request to the *course generator*. The course generator is responsible for choosing and arranging the content to be learned. The course generator contacts the *mathematical knowledge base* in order to fetch the identifiers (IDs) of the mathematical concepts that are required for understanding the goal concepts, queries the student model in order to find out about the user's prior knowledge and preferences, and uses *pedagogical rules* to select, annotate, and arrange the content – including examples and exercises – in a way that is suitable for the learner. The resulting instructional graph, a list of IDs, is sent to the *presentation engine* that retrieves the actual mathematical content corresponding to the IDs and that transforms the `XML`-data to output-pages which are then presented via the user's browser.

The *course* generator and the suggestion mechanism [10] work with the rule-based system Jess [6] that evaluates the (pedagogical) rules in order to decide which particular adaptation and content to select and which actions to suggest. Jess uses the Rete algorithm [5] for optimization.

External systems such as the computer algebra systems Maple and MuPad and the proof planner Multi are integrated with ActiveMath.

They serve as cognitive tools [9] and support the learner in performing complex interactive exercises and they assist in producing feedback by evaluating the learner's input.Also, a diagnosis is passed to the student model in order to update the model.

In these exercises, ActiveMath does not necessarily guide the user strictly along a predefined expert solution. It may only evaluate whether the student's input is mathematically equivalent to an admissible subgoal, i.e., maybe irrelevant but not outside the solution space (see [3]). Moreover, the external systems can support the user by automated problem solving, i.e., they can take over certain parts in the problem solving process and thereby help the user to focus on certain learning tasks and to delegate routine tasks.

Actually, most diagnoses are known to be AI-hard problems. Most ITSs encode the possible problem solving steps and the most typical misconceptions into their solution space or into systems that execute them. From this encoding, the system diagnoses the misconception of a student. This is, however, (1) infeasible in realistic applications with large solution spaces and (2) it is in general impossible to represent all potential misconceptions of a student [17].

The *presentation engine* generates personalized web pages based on two frameworks: Maverick and Velocity. Maverick[2] is a minimalist MVC framework for web publishing using Java and J2EE, focusing solely on MVC logic. It provides a wiring between URLs, Java controller classes and view templates.

The presentation engine is a reusable component that takes a structure of `OMDoc`s and transforms them into a presentation output that can be PDF (print format) or `HTML` with different maths-presentations – Unicode or `MathML` – (screen format) [7]. Basically, the presentation pipeline comprises two stages: stage 1 encompasses Fetching, Pre-Processing and Transformation, while stage 2 consists of Assembly, Personalization and optional Compilation. Stage 1 deals with individual content fragments or items, which are written in `OMDoc` and stored in a knowledge base. At this stage, content items do not depend on the user who is to view them. They have unique identifiers and can be handled separately. It is only in stage 2 that items are composed to user-specific pages.

[2] Maverick: `http://mav.sourceforge.net/`

3 Adaptivity

ActiveMath adapts its course generation (and presentation) to the student's

- technical equipment (customization)
- environment variables, e.g., curriculum, language, field of study (contextualization) and
- her cognitive and educational needs and preferences such as learning goals, and prerequisite knowledge (personalization).

As for personalization, individual preferences (such as the style of presentation), goal-competencies, and mastery-level are considered by the course generator. On the one hand, the goal-competencies are characterized by concepts that are to be learned and on the other hand, by the competency-level to be achieved: knowledge (k), comprehension (c), or application (a). The learner can initialize her student model by self-assessment of her mastery-level of concepts and choose her learning goals and learning scenario, for instance, the preparation for an exam or learning from scratch for k-competency level. The course generator processes this information and updates the student model and generates pages/sessions as depicted in the screenshots of Figure 2 and 3. These two screenshots differ in the underlying scenarios as the captions indicate.

Adaptation to the capabilities of the learner occurs in course generation as well as in the suggestion mechanism. The course generation checks whether the mastery-level of prerequisite concepts is sufficient for the goal competency. If not, it presents the missing concepts and/or explanations, examples, exercises for these concepts to the learner when a new session is requested. The suggestion mechanism acts dynamically in response to the student's activities. Essentially, this mechanism works with two blackboards, a diagnosis blackboard and a suggestion blackboard on which particular knowledge sources operate.

We also investigated special scenarios that support a student's metacognitive activities, such as those proposed in the seminal book 'How to Solve it' by Polya [15]. A Polya-scenario structures the problem solutions by introducing headlines such as "understand the problem", "make a plan", "execute the plan", and "look back at the solution". It augments and structures exercises with additional prompts similar to the above headlines [12].

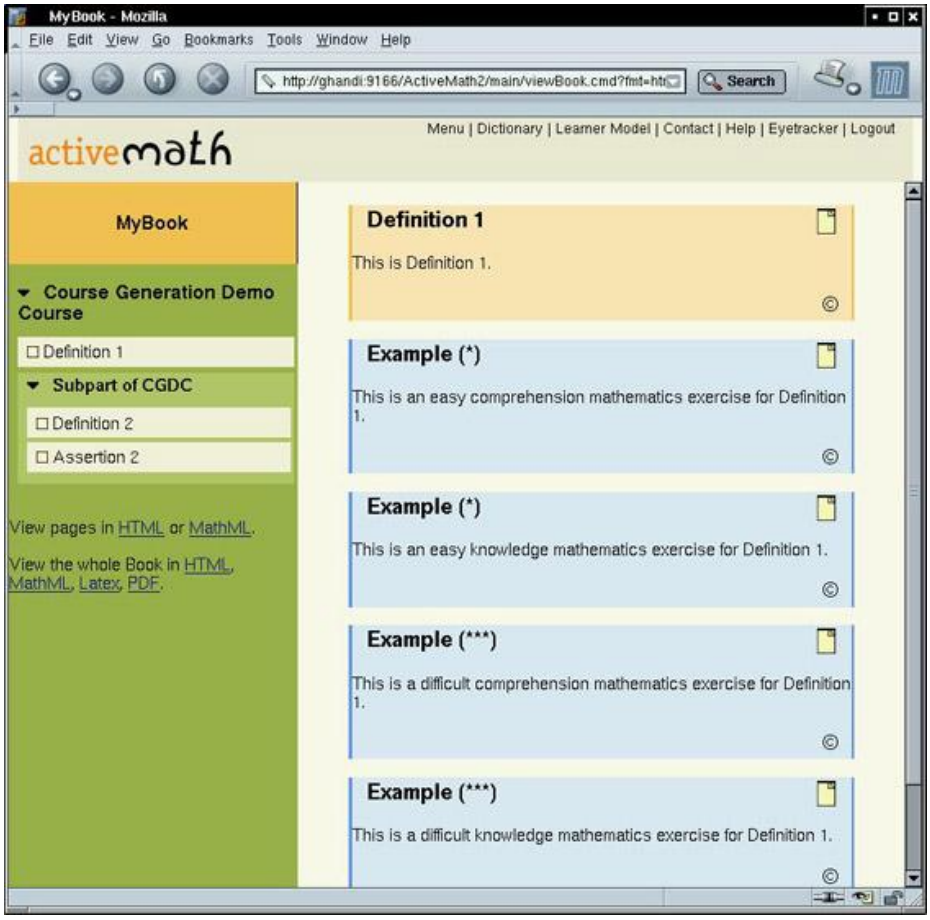

Fig. 2. A screen shot of an ActiveMath session for exam preparation

4 Student Modeling

User modeling has been a research area in AI for a long time. Actually, it started with early student modeling and still continues with the investigation of representational issues as well as diagnostic and updating techniques.

As ActiveMath' presentation is user-adaptive, it needs to incorporate persistent information about the user as well as a representation of the user's learning progress. Therefore, 'static' (wrt. the current session) properties such as field, scenario, goal concepts, and preferences as well as the 'dynamic' properties such as the mastery values for concepts and the student's actual behavior, are stored in the student model. These dif-

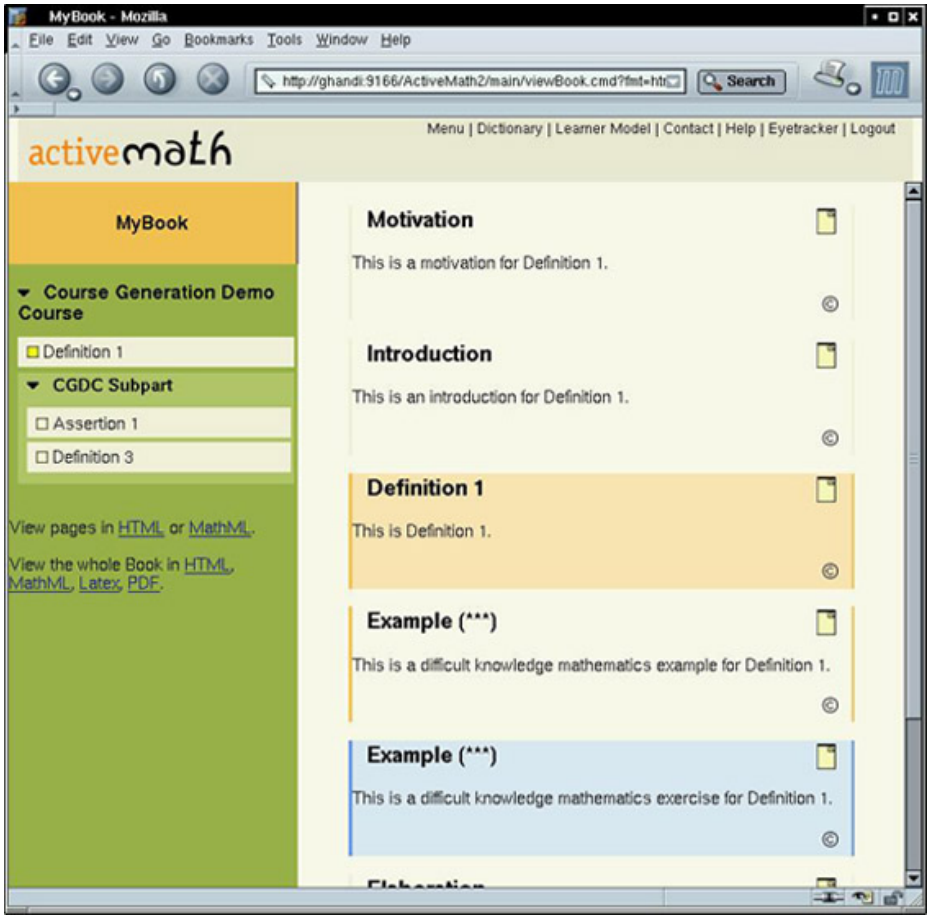

Fig. 3. k-level session of ActiveMath

ferent types of information are stored separately in the *history* and the static and dynamic *profiles*.

The profile is initialized with the learner's entries submitted to ActiveMath' registration page which describe the preferences (static), scenario, goals (static for the current session), and self-assessment values for knowledge, comprehension, and application of concepts (dynamic).

The history component stores the information about the learner's actions. Its elements contain information such as the IDs of the content of a read page or the ID of an exercise, the reading time, and the success rate of the exercise. Meanwhile, we developed a "poor man's eye-tracker" which allows to trace the student's attention and reading time in detail.

To represent the concept mastery assessment, the current (dynamic) profile contains values for a subset of the competences of Bloom's mastery taxonomy [2]:

- Knowledge
- Comprehension
- Application.

Finishing an exercise or going to another page triggers an updating process of the student model. Since different types of learner actions can exhibit different competencies, reading a concept mainly updates 'knowledge' values, following examples mainly updates 'comprehension', and solving exercises mainly updates 'application'. When the student model receives the notification that a student has finished reading a page, an evaluator fetches the list of its items and their types (concept, example, ...) and delivers an update of the values of those items. When the learner finishes an exercise, an appropriate evaluator delivers an update of the values of the involved concepts that depends on the difficulty and on the rating of how successful the solution was.

The student model is inspectable and modifiable by the student as shown in Figure 4. Our experience is that students tend to inspect their student model in order to plan what to learn next.

5 Knowledge Representation

As opposed to the purely syntactic representation formats for mathematical knowledge such as LaTex or `HTML`, the knowledge representation used by ActiveMath is the *semantic* `XML`-language `OMDoc` which is an extension of `OpenMath` [4]. `OpenMath` provides a collection of `OpenMath` objects together with a grammar for the representation of mathematical objects and sets of standardized symbols (the content-dictionaries). That is, `OpenMath` talks about objects rather than syntax.

Since `OpenMath` does not have any means to represent the content of a mathematical *document* nor its structure, `OMDoc` defines logical units such as "definition", "theorem", and "proof". In addition, the purely mathematical `OMDoc` is augmented by educational metadata such as difficulty of a learning object or type of an exercise.

This representation has several advantages, among them

- it is human and machine understandable
- the presentation of mathematical objects can in principle be copied and pasted

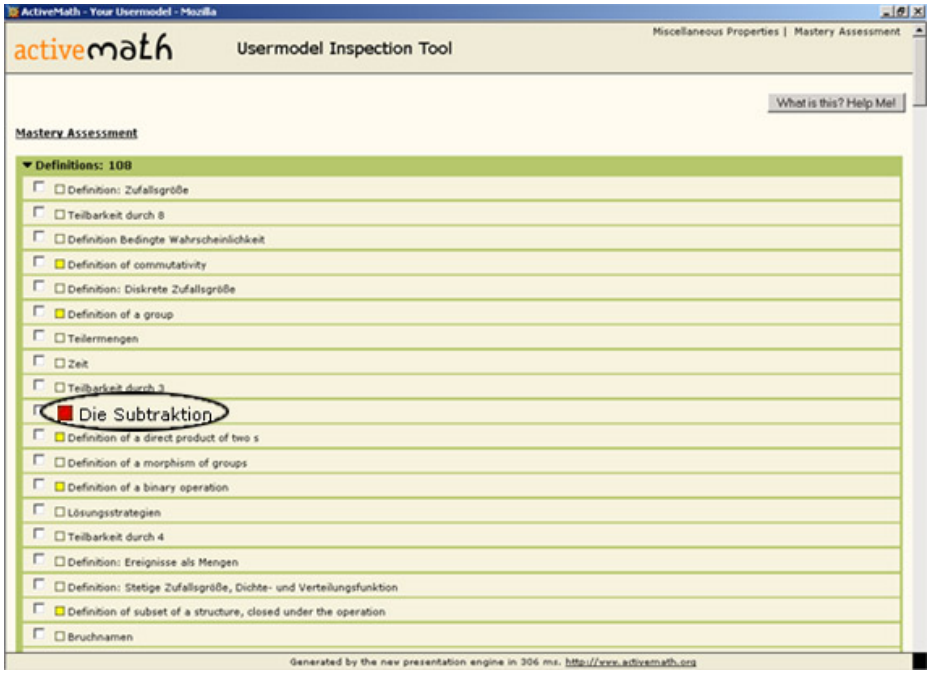

Fig. 4. Inspection of the student model (mastery-level)

– the presentations can automatically and dynamically be linked to concepts and learning objects and thus, found by ActiveMath's dictionary when clicking on a concept or formula in the course.
For more details, see [11].

6 Conclusion, Related and Future Work

The intelligent tutoring systems group – at the DFKI Saarbrücken and at the University of Saarland – has been developing the web-based ITS ActiveMath now for several years. A demo (and demo guide) is available at `http://www.activemath.org`.

This system is configurable with pedagogical strategies, content, and presentational style sheets as well as with external problem solving systems. It employs a number of AI-techniques to realize adaptive course generation, student modeling, feedback, interactive exercises, and a knowledge representation that is expedient for the semantic Web.

Related Work Most web-based learning systems (particularly commercial ones) offer fixed multimedia web pages and facilities for user man-